symmetry

MDPI

Article

The Root Extraction Problem for Generic Braids

María Cumplido [1,2,*], Juan González-Meneses [3,*] and Marithania Silvero [4,*]

1 Institut de Mathématiques de Bourgogne, UMR 5584, CNRS, Univ. Bourgogne Franche-Comté, 21000 Dijon, France
2 Department of Mathematics, Heriot-Watt University, Edinburgh, Scotland EH14 4AS, UK
3 Departamento de Álgebra, Universidad de Sevilla, 41012 Sevilla, Spain
4 Departamento de Ciencias Integradas, Universidad de Huelva, 21007 Huelva, Spain
* Correspondence: M.Cumplido@hw.ac.uk (M.C.); meneses@us.es (J.G.-M.); marithania@us.es (M.S.)

Received: 24 September 2019; Accepted: 17 October 2019; Published: 23 October 2019

Abstract: We show that, generically, finding the k-th root of a braid is very fast. More precisely, we provide an algorithm which, given a braid x on n strands and canonical length l, and an integer $k > 1$, computes a k-th root of x, if it exists, or guarantees that such a root does not exist. The generic-case complexity of this algorithm is $O(l(l+n)n^3 \log n)$. The non-generic cases are treated using a previously known algorithm by Sang-Jin Lee. This algorithm uses the fact that the ultra summit set of a braid is, generically, very small and symmetric (through conjugation by the Garside element Δ), consisting of either a single orbit conjugated to itself by Δ or two orbits conjugated to each other by Δ.

Keywords: braid groups; algorithms in groups; group-based cryptography

1. Introduction

Group theory is 'the language of symmetry', as it is beautifully explained by Marcus du Sautoy in his book *Symmetry*. In this paper we will deal with a fascinating family of groups discovered by Emil Artin: Braid groups.

There are several computational problems in braid groups that have been proposed for their potential applications in cryptography [1]. Initially, the conjugacy problem in the braid group $\mathbb{B}_n$ was proposed as a non-commutative alternative to the discrete logarithm problem [2,3]. Later, some other problems were proposed, including the k-th root extraction problem: given $x \in \mathbb{B}_n$ and an integer $k > 1$, find $a \in \mathbb{B}_n$ such that $a^k = x$.

The interest of braid groups for cryptography has decreased considerably, mainly due to the appearance of algorithms which solve the conjugacy problem extremely fast in the generic case [4–6]. The main problem with the proposed cryptographic protocols turns out to be the key generation. Public and secret keys are chosen 'at random', and this implies that the protocols are insecure against algorithms which have a fast generic-case complexity.

While the future of braid-cryptography depends on finding a good key-generation procedure, there are some other problems in braid groups whose generic-case complexity is still to be studied. This is the case of the k-th root (extraction) problem.

A priori, the study of the generic case for the k-th root problem could be though to be nonsense as, generically, the k-th root of a braid x does not exist. But we should think of the braid x as the k-th power of a generic braid: in protocols based on this problem, a secret braid a is chosen at random, and the braid $x = a^k$ is made public. Hence we are dealing with braids for which a k-th root is known to exist. In any

case, the algorithm in this paper not only shows that root extraction in braid groups is generically very fast, but can also be used by those mathematicians needing a simple algorithm for finding a k-th root of a braid (or proving that it does not exist), which works in most cases.

There are already known algorithms to solve the k-th root problem in braid groups and, more generally, in Garside groups [7,8]. But these algorithms can be simplified a lot in the generic case, as we will show in this paper.

The plan of this paper is as follows. In Section 2 we provide the necessary tools to describe the situation and attack the problem. Then in Section 3, we prove the theoretical results needed for our proposed algorithm, which is given in Section 4, together with the study of its generic-case complexity.

This generic-case complexity turns out to be quadratic on the canonical length l of the braid, if the number n of strands is fixed. More precisely, the generic-case complexity is $O(l(l+n)n^3 \log n)$ (Theorem 6).

2. Preliminaries

2.1. Garside Structure of $\mathbb{B}_n$

A group G is said to be a *Garside group* [9] if it admits a submonoid $\mathcal{P}$ (whose elements are called *positive*) such that $\mathcal{P} \cap \mathcal{P}^{-1} = \{1\}$, and a special element $\Delta \in \mathcal{P}$, called *Garside element*, satisfying the following properties:

- The partial order $\preccurlyeq$ in G defined by $a \preccurlyeq b$ if $a^{-1}b \in \mathcal{P}$ is a lattice order. If $a \preccurlyeq b$ we say that a is a *prefix* of b. The lattice structure implies that for all $a, b \in G$ there exists a unique meet $a \wedge b$ and a unique join $a \vee b$ with respect to $\preccurlyeq$. Notice that this partial order is invariant under left-multiplication.
- The set of simple elements $\mathcal{S} := \{s \in G \mid 1 \preccurlyeq s \preccurlyeq \Delta\}$ is finite and generates G.
- Conjugation by Δ preserves $\mathcal{P}$, that is, $\Delta^{-1}\mathcal{P}\Delta = \mathcal{P}$.
- $\mathcal{P}$ is atomic: the atoms are the indivisible elements of $\mathcal{P}$ (elements $a \in \mathcal{P}$ for which there is no decomposition $a = bc$ with non-trivial elements $b, c \in \mathcal{P}$). Then, for every $x \in \mathcal{P}$ there is an upper bound on the number of atoms in a decomposition of the form $x = a_1 a_2 \cdots a_n$, where each a_i is an atom.

One of the main examples of Garside groups is the braid group on n strands, denoted by $\mathbb{B}_n$. This group has a standard presentation due to Artin [10]:

$$\mathbb{B}_n = \left\langle \sigma_1, \sigma_2, ..., \sigma_{n-1} \;\middle|\; \begin{array}{ll} \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j & \text{if } |i-j| = 1 \\ \sigma_i\sigma_j = \sigma_j\sigma_i & \text{if } |i-j| > 1 \end{array} \right\rangle.$$

Attending to the above presentation, a braid is said to be positive if it can be written as a product of positive powers of the generators $\{\sigma_i\}_{i=1}^n$. The set of positive braids forms the monoid $\mathcal{P}$ corresponding to the classical Garside structure of $\mathbb{B}_n$. We will denote this monoid by $\mathbb{B}_n^+$.

The usual Garside element in $\mathbb{B}_n^+$, which we denote Δ_n, is defined recursively setting $\Delta_2 = \sigma_1$ and

$$\Delta_n = \Delta_{n-1}\sigma_{n-1}\sigma_{n-2}\cdots\sigma_1,$$

for all $n > 2$. We will often write Δ and omit the subindex n when there is no ambiguity.

Consider now the inner automorphism $\tau : \mathbb{B}_n \to \mathbb{B}_n$ determined by Δ. That is, $\tau(x) = \Delta^{-1}x\Delta$. One can easily show from the presentation of $\mathbb{B}_n$ that $\tau(\sigma_i) = \sigma_{n-i}$ for $1 \leq i \leq n-1$. Hence τ has order 2 and Δ^2 is central. In fact, the center of $\mathbb{B}_n$ is cyclic, generated by Δ^2 [11].

The set $\mathcal{S}$ of simple elements and the automorphism τ will be very important in the sequel.

2.2. Normal Forms, Cyclings and Decyclings

It is well-known that Garside groups have solvable word problem, as one can compute a normal form for each element.

Let us first define the *right complement* of a simple element $s \in \mathcal{S}$ as $\partial(s) = s^{-1}\Delta$. That is, $\partial(s)$ is the only element $t \in \mathcal{P}$ such that $st = \Delta$. Let us see that $\partial(s) = t$ is also a simple element. Recall that the simple elements are the positive prefixes of Δ. Since τ preserves $\mathcal{P}$ (by definition of Garside group), we have that $\tau(s)$ is positive. Now

$$st\tau(s) = \Delta\tau(s) = s\Delta,$$

hence $t\tau(s) = \Delta$, which implies that t is a positive prefix of Δ, that is, $t \in \mathcal{S}$. It follows that we have a map $\partial : \mathcal{S} \to \mathcal{S}$. Notice that, by definition, $\partial^2 \equiv \tau$.

Given two simple elements $s, t \in \mathcal{S}$, we say that the decomposition st is *left weighted* if s is the biggest possible simple element (with respect to $\preccurlyeq$) in any decomposition of the element st as a product of two simple elements. This condition can be restated as $\partial(s) \wedge t = 1$, i.e., $\partial(s)$ and t have no non-trivial prefixes in common.

Definition 1 ([12,13]). *The left normal form of an element $x \in \mathbb{B}_n$ is the unique decomposition $x = \Delta^p x_1 \cdots x_l$ so that $p \in \mathbb{Z}$, $l \geq 0$, $x_i \in \mathcal{S} \setminus \{1, \Delta\}$ for $i = 1, \ldots, l$, and $x_i x_{i+1}$ is a left weighted decomposition, for $i = 1, \ldots, l-1$.*

Given such a decomposition, we define the *infimum*, *supremum* and *canonical length* of x as $\inf(x) = p$, $\sup(x) = p + l$ and $\ell(x) = l$, respectively. Equivalently, the infimum and supremum of x can be defined as the maximum and minimum integers p and s so that $\Delta^p \preccurlyeq x \preccurlyeq \Delta^s$ (see [12]).

It is important to notice that conjugation by Δ preserves the Garside structure of $\mathbb{B}_n$. Hence, if the left normal form of a braid x is $\Delta^p x_1 \cdots x_l$, then the left normal form of $\tau(x)$ is $\Delta^p \tau(x_1) \cdots \tau(x_l)$. We will make use of this property later.

Garside groups also have solvable conjugacy problem. One of the main tools to solve problems related to conjugacy in braid groups are the summit sets, which are subsets of the conjugacy class of a braid. Throughout this article we are going to use two of them: the *super summit set* [12] and the *ultra summit set* [4]. Let us first introduce some concepts:

Definition 2. *Let $x = \Delta^p x_1 \cdots x_l$ be in left normal form, with $l > 0$. Notice that we can write:*

$$x = \tau^{-p}(x_1)\Delta^p x_2 \cdots x_l.$$

We define the initial factor of x as $\iota(x) = \tau^{-p}(x_1)$, and the final factor of x as $\varphi(x) = x_l$. We can then write:

$$x = \iota(x)\Delta^p x_2 \cdots x_l \quad \text{and} \quad x = \Delta^p x_1 \cdots x_{l-1}\varphi(x).$$

If $l = 0$, we set $\iota(x) = 1$ and $\varphi(x) = \Delta$.

Notice that, as τ^2 is the identity, we actually have either $\iota(x) = x_1$ if p is even, or $\iota(x) = \tau(x_1)$ if p is odd. This happens in braid groups, but not in other Garside groups in which the order of τ is bigger.

Definition 3 ([12]). *Let $x = \Delta^p x_1 \cdots x_l$ be in left normal form, with $l > 0$. The cycling and decycling of x are the conjugates of x defined, respectively, as*

$$\mathbf{c}(x) = \Delta^p x_2 \cdots x_l \iota(x) \quad \text{and} \quad \mathbf{d}(x) = \varphi(x)\Delta^p x_1 \cdots x_{l-1}.$$

Thus $\mathbf{c}(x)$ is the conjugate of x by $\iota(x)$, and that $\mathbf{d}(x)$ is the conjugate of x by $\varphi(x)^{-1}$.

Cyclings and decyclings were defined in [12] in order to try to simplify the braid x by conjugations. Usually, if $l \geq 2$, the decomposition $\Delta^p x_2 \cdots x_l \iota(x)$ is **not** the left normal form of $\mathbf{c}(x)$. So $\mathbf{c}(x)$ could a priori have a shorter normal form (with less factors). A similar situation happens for $\mathbf{d}(x)$.

If $\Delta^p x_2 \cdots x_l \iota(x)$ is actually the left normal form of $\mathbf{c}(x)$ (when $l \geq 2$), we say that the braid x is *rigid*. This happens if and only if $x_l \iota(x)$ (that is, $\varphi(x)\iota(x)$) is a left weighted decomposition. We can extend this definition to every case, when $l \geq 0$:

Definition 4. *We say that $x \in \mathbb{B}_n$ is rigid if $\varphi(x)\iota(x)$ is a left weighted decomposition.*

If x is rigid, neither cycling nor decycling can simplify its normal form $x = \Delta^p x_1 \cdots x_l$. Actually, the normal forms of the iterated cyclings of x are, if p is even:

$$\mathbf{c}(x) = \Delta^p x_2 \cdots x_l x_1, \qquad \mathbf{c}^2(x) = \Delta^p x_3 \cdots x_l x_1 x_2, \qquad \ldots$$

so $\mathbf{c}^l(x) = x$ in this case. In the case when p is odd we have:

$$\mathbf{c}(x) = \Delta^p x_2 \cdots x_l \tau(x_1), \qquad \mathbf{c}^2(x) = \Delta^p x_3 \cdots x_l \tau(x_1)\tau(x_2), \qquad \ldots$$

so $\mathbf{c}^{2l}(x) = x$ in this case.

In the same way, if x is rigid we have, for p even:

$$\mathbf{d}(x) = \Delta^p x_l x_1 \cdots x_{l-1}, \qquad \mathbf{d}^2(x) = \Delta^p x_{l-1} x_l x_1 \cdots x_{l-2}, \qquad \ldots$$

so $\mathbf{d}^l(x) = x$ in this case. If p is odd we get:

$$\mathbf{d}(x) = \Delta^p \tau(x_l) x_1 \cdots x_{l-1}, \qquad \mathbf{d}^2(x) = \Delta^p \tau(x_{l-1})\tau(x_l) x_1 \cdots x_{l-2}, \qquad \ldots$$

so $\mathbf{d}^{2l}(x) = x$ in this case. We then see that, if x is rigid, iterated cyclings and decyclings correspond to cyclic permutations of the factors in the normal form of x (possibly conjugated by Δ, if p is odd); moreover, when applied to rigid braids, $\mathbf{c}$ and $\mathbf{d}$ are inverses of each other.

2.3. Summit Sets

Let now $x \in \mathbb{B}_n$ be an arbitrary braid (not necessarily rigid). Consider the conjugacy class of x, denoted $x^{\mathbb{B}_n}$, and write $\inf_s(x)$ (resp. $\sup_s(x)$) for the maximal infimum (resp. the minimal supremum) of an element in $x^{\mathbb{B}_n}$. These numbers are known to exist [12], and are called the *summit infimum* and the *summit supremum* of x, respectively. Set $\ell_s(x) = \sup_s(x) - \inf_s(x)$, the *summit length* of x. It is shown in [12] that the elements in $x^{\mathbb{B}_n}$ having the shortest possible normal form are those whose canonical length is precisely $\ell_s(x)$, and they coincide with the elements whose infimum and supremum are equal to $\inf_s(x)$ and $\sup_s(x)$, respectively. The set formed by these elements is called the *supper summit set* of the braid x:

$$SSS(x) = \left\{ y \in x^{\mathbb{B}_n} \mid \ell(y) = \ell_s(x) \right\} = \left\{ y \in x^{\mathbb{B}_n} \mid \inf(y) = \inf_s(x),\ \sup(y) = \sup_s(x) \right\}.$$

Starting from x, it is possible to obtain an element in $SSS(x)$ by applying cyclings and decyclings iteratively. It is known [12] that if $\inf(x) < \inf_s(x)$ then the infimum of x can be increased by iterated cycling. Actually, in this case $\inf(x) < \inf(\mathbf{c}^k(x))$ for some $k < \frac{n(n-1)}{2}$ (see [14]). Hence, every $\frac{n(n-1)}{2}$ iterations either the infimum has increased, or one is sure to have an element whose infimum is the summit infimum.

In the same way, if $\sup(x) > \sup_s(x)$, then the supremum of x can be decreased by iterated decycling [12], and in that case $\sup(x) > \sup(\mathbf{d}^k(x))$ for some $k < \frac{n(n-1)}{2}$ [14]. Hence, every $\frac{n(n-1)}{2}$ iterations either the supremum has decreased, or we are sure to have an element whose supremum is the summit supremum. Since decycling can never decrease the infimum of an element, it follows that starting with any $x \in \mathbb{B}_n$ and applying iterated cycling (until summit infimum is obtained) followed by iterated decycling (until summit supremum is obtained) yields an element $y \in SSS(x)$.

The super summit set $SSS(x)$ is a finite set, but it is usually huge, so smaller subsets of the conjugacy class of x were defined in order to solve the conjugacy problem of x more efficiently. Namely, the *ultra summit set* of x, denoted by $USS(x)$, is a subset of $SSS(x)$ defined as follows [4]:

$$USS(x) = \{y \in SSS(x) \mid \mathbf{c}^m(y) = y \text{ for some } m > 0\}.$$

Since $SSS(x)$ is finite, the subset $USS(x)$ is also finite. It is then clear that one obtains an element is $USS(x)$ by iterated application of cycling, starting from an element in $SSS(x)$, when a repeated element is obtained. Actually, the whole orbit under cycling of an element in $USS(x)$ belongs to $USS(x)$. So $USS(x)$ is a finite set of orbits under cycling.

Notice that every rigid braid belongs to its ultra summit set, as cylings and decyclings are basically cyclic permutations of its factors. It is shown in [15] that, if x is conjugate to a rigid braid and $\ell_s(x) > 1$, then $USS(x)$ coincides with the set of rigid conjugates of x.

There is actually a simpler way, in the general case, to obtain an element in $USS(x)$ starting from x. Instead of using cyclings and decyclings, one can use the following single type of conjugation:

Definition 5 ([5]). *Given $x \in \mathbb{B}_n$, the cyclic sliding of x is defined as $\mathfrak{s}(x) = \mathfrak{p}(x)^{-1} x\, \mathfrak{p}(x)$, where $\mathfrak{p}(x) = \iota(x) \wedge \partial(\varphi(x))$.*

Theorem 1 ([5]). *Given $x \in \mathbb{B}_n$, there are integers $0 \leq k < t$ such that $\mathfrak{s}^k(x) = \mathfrak{s}^t(x)$. For every such pair of integers, one has $\mathfrak{s}^k(x) \in USS(x)$.*

By the above result, one can obtain an element in $USS(x)$ by iterated cyclic sliding starting form x. Furthermore, if x is conjugate to a rigid element (this will be the generic situation, as we will see in Section 2.4), iterated cyclic sliding yields the *shortest* positive conjugating element from x to a rigid element.

Theorem 2 ([5]). *Let $x \in \mathbb{B}_n$ and suppose that x is conjugate to a rigid braid. Then there is an integer $k > 0$ such that $\mathfrak{s}^k(x)$ is rigid. Moreover, the conjugating element α from x to $\mathfrak{s}^k(x)$, that is,*

$$\alpha = \mathfrak{p}(x)\, \mathfrak{p}(\mathfrak{s}(x))\, \mathfrak{p}(\mathfrak{s}^2(x)) \cdots \mathfrak{p}(\mathfrak{s}^{k-1}(x))$$

is the smallest positive element (with respect to $\preccurlyeq$) conjugating x to a rigid element, meaning that for every positive element β such that $\beta^{-1} x \beta$ is rigid, one has $\alpha \preccurlyeq \beta$.

After obtaining one element in $USS(x)$, it is possible to compute all elements in $USS(x)$ together with conjugating elements connecting them. In this way, one solves the conjugacy problem in $\mathbb{B}_n$, as two elements x and y are conjugate if and only if $USS(x) = USS(y)$ or, equivalently, if $USS(x) \cap USS(y) \neq \emptyset$. Then, in order to check whether x and y are conjugate, one can compute the whole set $USS(x)$, and one element $\widetilde{y} \in USS(y)$. Then, x and y are conjugate if and only if $\widetilde{y} \in USS(x)$. By construction, one can even compute a conjugating element from x to y.

In order to understand the forthcoming proofs in this paper, we will need to describe some conjugating elements connecting the elements of $USS(x)$.

Definition 6 ([4]). *Let $x \in \mathbb{B}_n$ and $y \in USS(x)$. A simple non-trivial element $s \in \mathcal{S}$ is said to be a* minimal simple element *for y if $y^s \in USS(x)$ and $y^t \notin USS(x)$, for every $1 \prec t \prec s$.*

In [4], Gebhardt showed that for any two elements $y, z \in USS(x)$ there exists a sequence

$$y = y_1 \xrightarrow{c_1} y_2 \xrightarrow{c_2} \cdots \to y_t \xrightarrow{c_t} y_{t+1} = z,$$

where c_i is a minimal simple element for y_i, and $y_{i+1} = c_i^{-1} y_i c_i$, for $i = 1, \ldots, t$. Moreover, he introduced an algorithm to compute all minimal simple elements for a given $y \in USS(x)$. This allows to construct a directed graph Γ_x, whose vertices correspond to elements of $USS(x)$, and whose arrows correspond to minimal simple elements, in such a way that for every minimal simple element s for y, there is an edge with label s from y to $y^s = s^{-1} y s$. By the above discussion, it follows that Γ_x is a connected graph, and this is why $USS(x)$ can be computed starting with a single vertex, iteratively computing the minimal simple elements corresponding to each known vertex, until all vertices are obtained.

We will later see that, generically, ultra summit sets are really small. Actually, they usually have a very simple structure, that we explain now.

Lemma 1 ([16]). *Let $y \in USS(x)$ with $\ell(y) > 0$ and let s be a minimal simple element for y. Then, s is a prefix of either $\iota(y)$ or $\partial(\varphi(y))$, or both.*

The above lemma allows us to classify the arrows in Γ_x into two groups: a directed edge labelled by s starting at $y \in USS(x)$ is black (resp. grey), if s is a prefix of $\iota(x)$ (resp. of $\partial(\varphi(y))$). In principle, an edge could be of both colors at the same time (a bi-colored arrow, whose label is a prefix of both $\iota(x)$ and $\partial(\varphi(x))$), but not in the case of rigid braids, as $\iota(x) \wedge \partial(\varphi(x)) = 1$ if x is rigid. Actually, this is a necessary and sufficient condition:

Lemma 2 ([16]). *A braid $y \in USS(x)$ with $\ell(y) > 0$ is rigid if and only if none of the edges starting at y is bi-colored.*

Definition 7. *Given a braid $x \in \mathbb{B}_n$, its associated $USS(x)$ is* minimal *if $\ell_s(x) > 1$ and, for every vertex y in the graph Γ_x, there are exactly two directed edges starting at y, a black one labeled $\iota(y)$ and a grey one labeled $\partial(\varphi(y))$.*

Notice that, as a consequence of Lemma 2, if $USS(x)$ is minimal then all elements in $USS(x)$ are rigid. Moreover, the arrow labeled $\iota(y)$ corresponds to a cycling of y, and the arrow labeled $\partial(\varphi(y))$ corresponds to a *twisted decycling* of y, meaning a decycling followed by the automorphism τ. This implies that, if $USS(x)$ is minimal, the elements of $USS(x)$ are obtained from y by applying $\mathbf{c}$ and $\tau \circ \mathbf{d}$ in every possible way. Since y is rigid, cyclings and decyclings basically correspond to cyclic permutations of the factors. Therefore, if $USS(x)$ is minimal, it consists of either two orbits under cycling (conjugate to each other by Δ), or one orbit under cycling (conjugate to itself by Δ). If the infimum of y is even, the orbit of y has at most $\ell(y) = \ell_s(x) \leq \ell(x)$ elements, so the size of $USS(x)$ is at most $2\ell(x)$. If the infimum of y is odd, the orbit of y has at most $2\ell(y) \leq 2\ell(x)$ elements, and it is conjugate to itself by Δ, so it is the only orbit. Therefore, in any case, if $USS(x)$ is minimal it has at most $2\ell(x)$ elements.

Remark 1. *In order to see whether $USS(x)$ is minimal, one should a priori check the condition in Definition 7 for every element in $USS(x)$. But it is actually shown in ([17], Theorem 4.6) that, given $y \in USS(x)$, the set $USS(x)$ is minimal if and only if $\ell(y) > 1$ and the minimal simple elements for y are precisely $\iota(y)$ and $\partial(\varphi(y))$. Hence, one just needs to compute the minimal elements for a single arbitrary element $y \in USS(x)$.*

Let us see that this case, in which $USS(x)$ is so small and has such a simple structure, is generic.

2.4. Generic Braids

Since $\mathbb{B}_n$ is an infinite set, it is necessary to explain what we mean by 'picking a random braid' or by saying that a braid is 'generic'. Even if we fix the subset of braids of a given length, we must specify if we choose braids from the subset with a uniform distribution, or if we pick braids by choosing a random walk in the Cayley graph, which are the two usual situations.

We will consider the Cayley graph of the braid group $\mathbb{B}_n$, taking as generators the simple braids, and assume that each edge of the Cayley graph has length 1, so it becomes a metric space. Let us point out that left normal forms of braids are closely related to geodesics in this Cayley graph [18].

Now let $B(r)$ denote the ball of radius r centered at the trivial braid 1. As the number of simple braids is finite, the set $B(r)$ is a finite subset of $\mathbb{B}_n$. We will consider the uniform distribution within this set. It turns out that 'most' elements in $B(r)$ have a very simple ultra summit set:

Theorem 3 ([17]). *The proportion of braids in $B(r)$ whose ultra summit set is minimal tends to* 1 *exponentially fast, as r tends to infinity.*

This is why we can say that the ultra summit set of a 'generic braid' is minimal. Moreover, the above result was obtained by refining the following theorem, which gives some important information concerning the elements in $B(r)$. We have simplified the statement to adapt it to our situation:

Theorem 4 ([19]). *The proportion of braids x in $B(r)$ which are conjugate to a rigid braid $y = \alpha^{-1}x\alpha$, in such a way that α is a positive braid with $\ell(\alpha) < \ell(x)$, tends to* 1 *exponentially fast, as r tends to infinity.*

Therefore, not only generic braids have minimal ultra summit sets (made of rigid braids), but one can also obtain a rigid conjugate of a generic braid x very fast, applying iterated cyclic sliding to x. By Theorem 2, the obtained conjugating element will be the smallest possible positive conjugator, so its canonical length will be smaller than $\ell(x)$. Once that a rigid conjugate y (which belongs to $USS(x)$) is obtained, one can compute the whole $USS(x)$ very fast, as it consists of at most $2\ell(x)$ elements, connected by cyclings and twisted decyclings. This is why solving the conjugacy problem in braid groups is generically very fast.

We will also be interested in the centralizer $Z(x)$ of a braid x. Notice that if $y = \alpha^{-1}x\alpha$, then $Z(y) = \alpha^{-1}Z(x)\alpha$. Therefore, knowing $Z(y)$ is equivalent to knowing $Z(x)$, via α. We will then be interested in $Z(y)$ for $y \in USS(x)$.

Definition 8. *Let $x \in \mathbb{B}_n$ and $y \in USS(x)$, and let t be the smallest positive integer such that $\mathbf{c}^t(y) = y$. Denote $p_i := \iota(\mathbf{c}^{i-1}(y))$ the positive element conjugating $\mathbf{c}^{i-1}(y)$ to $\mathbf{c}^i(y)$, for $i = 1, \ldots, t$. Then the preferred cycling conjugator of y is defined as*

$$PC(y) = p_1 p_2 \cdots p_t.$$

In other words, $PC(y)$ corresponds to the conjugating element along the whole cycling orbit of y. By construction, $PC(y)$ commutes with y.

In the generic case (when $USS(x)$ is minimal), it turns out that $Z(x)$ is isomorphic to $\mathbb{Z}^2$, and one can describe the generators of $Z(y)$ for any $y \in USS(x)$ (and thus of $Z(x)$) in a very explicit way:

Theorem 5 ([17]). *Let $x \in \mathbb{B}_n$ and $y \in USS(x)$. Let $PC(y) = p_1 \cdots p_t$ as above. If $USS(x)$ is minimal, then all elements in $USS(x)$ are rigid, $Z(x) \simeq Z(y) \simeq \mathbb{Z}^2$, and one of the following conditions holds:*

(i) *$USS(x)$ has two orbits under cycling, conjugate to each other by Δ, and $Z(y) = \langle \Delta^2, PC(y) \rangle$.*

(ii) *$USS(x)$ has one orbit under cycling, conjugate to itself by Δ, and:*

- *If $\tau(y) = y$, then $Z(y) = \langle \Delta, PC(y) \rangle$.*
- *If $\tau(y) \neq y$, then t is even and $Z(y) = \langle \Delta^2,\ p_1 \cdots p_{\frac{t}{2}} \Delta^{-1} \rangle$.*

3. k-th Root Problem

Now we come to the central problem in this paper: given $x \in \mathbb{B}_n$ and an integer $k > 1$, find a k-th root of x. In other words, we want to either find $a \in \mathbb{B}_n$ such that $a^k = x$, or show that such a braid does not exist.

Notice that if $a^k = x$ then a belongs to $Z(x)$, the centralizer of x. It is interesting to know that finding a single solution a to the k-th root equation is basically the same as finding all possible solutions, as the complete set of solutions coincides with the conjugacy class of a in $Z(x)$:

Proposition 1. *Let $a, x \in \mathbb{B}_n$ be such that $a^k = x$ for some integer $k > 1$. Then the set $\sqrt[k]{x}$ of k-th roots of x is precisely*

$$\sqrt[k]{x} = a^{Z(x)} = \left\{ b \in \mathbb{B}_n \mid\ b = u^{-1}au,\ u \in Z(x) \right\}.$$

Proof. In [20], the second author proved that the k-th root of a braid is unique, up to conjugacy. That is, if $a, b \in \mathbb{B}_n$ satisfy $a^k = b^k = x$, then $a = u^{-1}bu$ for some $u \in \mathbb{B}_n$. Then one has $x = b^k = u^{-1}a^k u = u^{-1}xu$, and hence $u \in Z(x)$. This proves that $\sqrt[k]{x} \subset a^{Z(x)}$.

On the other hand, if $b = a^{Z(x)}$ and we write $b = u^{-1}au$ for some $u \in Z(x)$, we have $b^k = u^{-1}a^k u = u^{-1}xu = x$, so $b \in \sqrt[k]{x}$. □

Observe that $a^k = x$ if and only if $(\alpha^{-1}a\alpha)^k = \alpha^{-1}x\alpha$ for any $\alpha \in \mathbb{B}_n$. Hence, given x, it suffices to solve the k-th root problem for any conjugate of x, for instance for some $y \in USS(x)$.

We will focus our attention in the generic case in which $USS(x)$ is minimal. Recall from Theorem 5 that in this case $Z(x) \simeq Z(y) \simeq \mathbb{Z}^2$. If we express the centralizer of y as $Z(y) = \langle v, w \rangle$, where v and w commute, we know that y has the form $y = v^c w^d$, for some $c, d \in \mathbb{Z}$ (and that this expression is unique, as any other expression would yield a different element of $Z(y)$). If we are able to express y in this way, then the k-th root problem is trivially solved:

Proposition 2. *Let $x \in \mathbb{B}_n$. Let $y \in USS(x)$ and suppose that $USS(x)$ is minimal. Let $Z(y) = \langle v, w \rangle$ and let $c, d \in \mathbb{Z}$ be such that $y = v^c w^d$. Then y admits a k-th root if and only if both c and d are multiples of k, and in this case the only k-th root of y is:*

$$a = v^{\frac{c}{k}} w^{\frac{d}{k}}.$$

Proof. We know from Theorem 5 that $Z(y) \simeq \mathbb{Z}^2$, so it is abelian. Hence, by Proposition 1, if a k-th root a of y exists then $\sqrt[k]{y} = a^{Z(y)} = \{a\}$. Therefore, if a k-th root exists, it is unique.

Suppose that the k-th root problem for y has a solution $a \in \mathbb{B}_n$. Then $a \in Z(y)$, and hence $a = v^r w^s$ for some $r, s \in \mathbb{Z}$. But since v and w commute, we have:

$$v^c w^d = y = a^k = (v^r w^s)^k = v^{rk} w^{sk}.$$

This implies that c and d are multiples of k, and that $a = v^r w^s = v^{\frac{c}{k}} w^{\frac{d}{k}}$.

Conversely, if c and d are multiples of k, we write $c = rk$ and $d = sk$ for some integers r, s, and we consider the element $a = v^r w^s$. Since v and w commute, it follows that $a^k = y$. □

By the above result, it follows that the only difficulty in solving the k-th root problem, in the generic case in which $USS(x)$ is minimal, is to express some $y \in USS(x)$ in terms of the generators of $Z(y)$. We know from Theorem 5 that there are three possible cases, depending on whether $USS(x)$ has two orbits under cycling, or has one orbit with $\tau(y) = y$, or has one orbit with $\tau(y) \neq y$. The three following results address each case:

Proposition 3. *Let $x \in \mathbb{B}_n$, and let $y = \Delta^p y_1 \cdots y_l \in USS(x)$, written in left normal form. Suppose that $USS(x)$ is minimal. Suppose also that $USS(x)$ has two orbits under cycling, conjugate to each other by Δ. Let $v = \Delta^2$ and $w = PC(y) = p_1 \cdots p_t$, so:*

$$Z(y) = \langle v, w \rangle = \langle \Delta^2, PC(y) \rangle.$$

If we write $c = p/2$ and $d = l/t$, then c and d are integers and we have: $y = v^c w^d$.

Proof. We know that, since $USS(x)$ is minimal, it consists of rigid elements. Hence iterated cycling corresponds to a cyclic permutation of the factors in the normal form of y (with possible conjugations by Δ, if p is odd).

Suppose that p is odd. Then $\mathbf{c}^l(y)$ is obtained from y by cyclically permuting all its l factors, conjugating all of them by Δ. Hence $\mathbf{c}^l(y) = \tau(y)$. This implies that $\tau(y) = \Delta^{-1} y \Delta$ is in the same orbit of y under cycling, but this is a contradiction with the hypotheses, as $USS(x)$ has two distinct orbits (the one containing y and the one containing $\tau(y)$). Therefore p is even.

Since p is even, iterated cyclings of y correspond exactly to cyclic permutations of the factors of y. By definition, t is the smallest positive integer such that $\mathbf{c}^t(y) = y$, and it is then clear that $\mathbf{c}^m(y) = y$ for some positive integer m if and only if m is a multiple of t. Since $\mathbf{c}^l(y) = y$, we finally obtain that l is a multiple of t. Then the normal form of y is as follows:

$$y = \Delta^p y_1 \cdots y_l = \Delta^p (y_1 \cdots y_t)(y_1 \cdots y_t) \cdots (y_1 \cdots y_t),$$

where $PC(y) = y_1 \cdots y_t$, and there are l/t parenthesized factors.

Now, if we write $c = p/2$ and $d = l/t$, these numbers are integers and we have:

$$v^c w^d = (\Delta^2)^c (PC(y))^d = \Delta^{2c} (y_1 \cdots y_t)^d = \Delta^p y_1 \cdots y_l = y.$$

□

Proposition 4. *Let $x \in \mathbb{B}_n$, and let $y = \Delta^p y_1 \cdots y_l \in USS(x)$, written in left normal form. Suppose that $USS(x)$ is minimal. Suppose also that $USS(x)$ has one orbit under cycling, conjugate to itself by Δ, and that $\tau(y) = y$. Let $v = \Delta$ and $w = PC(y) = p_1 \cdots p_t$, so:*

$$Z(y) = \langle v, w \rangle = \langle \Delta, PC(y) \rangle.$$

If we write $c = p$ and $d = l/t$, then c and d are integers and we have: $y = v^c w^d$.

Proof. We know that the left normal form of $\tau(y)$ is $\Delta^p \tau(y_1) \cdots \tau(y_l)$. Since $\tau(y) = y$, the normal forms of y and $\tau(y)$ must coincide, hence $\tau(y_i) = y_i$ for $i = 1, \ldots, l$.

This implies that iterated cyclings correspond to cyclic permutations of the factors of y. We do not care about the parity of p, as every factor of y is invariant under τ. It then follows that $PC(y) = y_1 \cdots y_t$, that t divides l and that the normal form of y is:

$$y = \Delta^p y_1 \cdots y_l = \Delta^p (y_1 \cdots y_t)(y_1 \cdots y_t) \cdots (y_1 \cdots y_t),$$

where there are l/t parenthesized factors.

Now, if we write $c = p$ and $d = l/t$, these numbers are integers and we have:

$$v^c w^d = \Delta^c (PC(y))^d = \Delta^c (y_1 \cdots y_t)^d = \Delta^p y_1 \cdots y_l = y.$$

□

Proposition 5. *Let $x \in \mathbb{B}_n$, and let $y = \Delta^p y_1 \cdots y_l \in USS(x)$, written in left normal form. Suppose that $USS(x)$ is minimal. Suppose also that $USS(x)$ has one orbit under cycling, conjugate to itself by Δ, and that $\tau(y) \neq y$. Let $v = \Delta^2$, $PC(y) = p_1 \cdots p_t$ and $w = p_1 \cdots p_{\frac{t}{2}} \Delta^{-1}$ (recall from Theorem 5 that t is even), so:*

$$Z(y) = \langle v, w \rangle = \langle \Delta,\ p_1 \cdots p_{\frac{t}{2}} \Delta^{-1} \rangle.$$

If we write $c = \frac{pt+2l}{2t}$ and $d = \frac{2l}{t}$, then c and d are integers and we have: $y = v^c w^d$.

Proof. We know from Theorem 5 that t is even, but let us see why this holds. We know that there exists some $m > 0$ so that $\tau(y) = \mathbf{c}^m(y)$; we take m as small as possible, and this implies that $\mathbf{c}^r(y) \neq y$ for $0 < r < m$. Now, it follows from their own definitions that τ and $\mathbf{c}$ commute, and therefore $y = \tau^2(y) = \tau(\mathbf{c}^m(y)) = \mathbf{c}^m(\tau(y)) = \mathbf{c}^{2m}(y)$. This implies that the length of the cycling orbit of y is a divisor of $2m$. It cannot be m (as $\mathbf{c}^m(y) = \tau(y) \neq y$), and it cannot be smaller than m (as $\mathbf{c}^r(y) \neq y$ for every $r < m$). Therefore, the length of the orbit is precisely $t = 2m$. The generators of $Z(y)$ are then $v = \Delta^2$ and $w = p_1 \cdots p_m \Delta^{-1}$.

We consider now two cases, depending on the parity of p. If p is even, since the first m cyclings transform y into $\tau(y)$, it follows that the left normal form of y is:

$$y = \Delta^p\ (y_1 \cdots y_m)\ (\tau(y_1) \cdots \tau(y_m))\ \cdots\ (y_1 \cdots y_m)\ (\tau(y_1) \cdots \tau(y_m)).$$

Then $l = 2rm$ for some positive integer r.

Recall that $PC(y)$ is the product of the first $t = 2m$ conjugating elements for cycling. The first m conjugating elements are $y_1, \ldots, y_m$, so $p_i = y_i$ for $i = 1, \ldots, m$. The following m conjugating elements are $\tau(y_1), \ldots, \tau(y_m)$. Hence, we have that

$$\begin{aligned} PC(y) &= p_1 \cdots p_t \\ &= y_1 \cdots y_m \tau(y_1) \cdots \tau(y_m) \\ &= y_1 \cdots y_m \ \tau(y_1 \cdots y_m) \\ &= p_1 \cdots p_m \Delta^{-1} p_1 \cdots p_m \Delta \\ &= \left(p_1 \cdots p_m \Delta^{-1}\right) \left(p_1 \cdots p_m \Delta^{-1}\right) \Delta^2 \\ &= w^2 v. \end{aligned}$$

Therefore, if p is even:

$$y = \Delta^p PC(y)^r = v^{\frac{p}{2}} \left(w^2 v\right)^r = v^{\frac{p}{2}+r} w^{2r} = v^c w^d,$$

where $c = \frac{pt+2l}{2t}$ and $d = \frac{2l}{t}$ (recall that $l = 2rm = rt$).

Consider now the case when p is odd. In this case, the left normal form of y is:

$$y = \Delta^p \ (y_1 \cdots y_m) \ (\tau(y_1) \cdots \tau(y_m)) \ \cdots \ (y_1 \cdots y_m) \ (\tau(y_1) \cdots \tau(y_m)) \ (y_1 \cdots y_m).$$

Then $l = (2r+1)m$ for some positive integer r.

As before, $PC(y)$ is the product of the first $t = 2m$ conjugating elements for cycling, but this time the first m conjugating elements for cycling are $\tau(y_1), \ldots, \tau(y_m)$, and therefore $p_i = \tau(y_i)$ for $i = 1, \ldots, m$. The following m conjugating elements are $y_1, \ldots, y_m$, so we have:

$$\begin{aligned} PC(y) &= p_1 \cdots p_t \\ &= \tau(y_1) \cdots \tau(y_m) y_1 \cdots y_m \\ &= p_1 \cdots p_m \ \tau(p_1 \cdots p_m) \\ &= p_1 \cdots p_m \Delta^{-1} p_1 \cdots p_m \Delta \\ &= \left(p_1 \cdots p_m \Delta^{-1}\right) \left(p_1 \cdots p_m \Delta^{-1}\right) \Delta^2 \\ &= w^2 v. \end{aligned}$$

Hence $PC(y) = w^2 v$ also when p is odd. Finally, we have:

$$\begin{aligned} y &= \Delta^p (y_1 \cdots y_m)(\tau(y_1) \cdots \tau(y_m)) \cdots (y_1 \cdots y_m)(\tau(y_1) \cdots \tau(y_m))(y_1 \cdots y_m) \\ &= (\tau(y_1) \cdots \tau(y_m))(y_1 \cdots y_m) \cdots (\tau(y_1) \cdots \tau(y_m))(y_1 \cdots y_m) \Delta^p (y_1 \cdots y_m) \\ &= PC(y)^r \Delta^p (y_1 \cdots y_m) \\ &= PC(y)^r \Delta^{p+1} \Delta^{-1} (y_1 \cdots y_m) \\ &= PC(y)^r \Delta^{p+1} (p_1 \cdots p_m) \Delta^{-1} \\ &= (w^2 v)^r v^{\frac{p+1}{2}} w \\ &= v^{\frac{2r+1+p}{2}} w^{2r+1} \\ &= v^c w^d, \end{aligned}$$

where $c = \frac{pt+2l}{2t}$ and $d = \frac{2l}{t}$ (recall that $2l = 2(2r+1)m = (2r+1)t$ in this case). □

4. An Algorithm to Find the k-th Root of a Braid

We end this paper by providing a detailed algorithm that summarizes the results from the previous section, together with a study of its complexity.

The results of the previous section are valid when $USS(x)$ is minimal (which is the generic case). In order to have an algorithm which always succeeds in finding the k-th root of a braid x, we need to include instructions on what to do if $USS(x)$ is not minimal. In those cases, one can use the algorithm in [7], which finds the k-th root of x in any case, considering the Garside group $G = \mathbb{Z} \ltimes (\mathbb{B}_n)^k$, where $\mathbb{Z} = \langle \delta \rangle$ acts on $(\mathbb{B}_n)^k$ by cyclic permutation of the coordinates. S. J. Lee shows that the braid x has a k-th root if and only if the ultra summit set of $\delta(x, 1, \ldots, 1)$ in G has an element of the form $\delta(h, \ldots, h)$. Hence, computing an ultra summit set in such a group also solves the root extraction problem in $\mathbb{B}_n$. It is not clear to us how big these ultra summit sets are in generic cases, while the algorithm presented in this paper is very simple, and generically very fast.

If one is not interested in programming the algorithm in [7], one could tell our algorithm to return 'fail' when $USS(x)$ is not minimal, obtaining an algorithm which will succeed only in the generic case. In any case, we present now the main result:

Theorem 6. *There is an algorithm that takes as input a braid $x = \Delta^p x_1 \ldots, x_l \in \mathbb{B}_n$ written in left normal form, and a positive integer $k > 1$, and finds a braid $a \in \mathbb{B}_n$ such that $a^k = x$, or guarantees that such a braid does not exist, whose generic-case complexity is $O(l(l+n)n^3 \log n)$.*

Proof. Algorithm 1, which uses the results from the previous section, constitutes a proof of the theorem. Let us describe it in detail.

The input is a braid $x = \Delta^p x_1 \cdots x_l \in \mathbb{B}_n$ in left normal form and an integer $k > 1$. First (lines 2–5), the algorithm applies iterated cyclic sliding to x, checking at each iteration whether the resulting braid y is rigid. As we will now see, if the algorithm applies cyclic sliding $l\left(\frac{n(n-1)}{2} - 1\right)$ times and no rigid braid is obtained, then we are not in the generic case stated in Theorem 4, hence the algorithm in [7] is applied. The number $l\left(\frac{n(n-1)}{2} - 1\right)$ is precisely l times the length of Δ minus one. Recall from Theorem 4 that in the generic case there is a positive element α conjugating x to a rigid braid, such that $\ell(\alpha) < \ell(x) = l$. If α is the smallest possible one, there is no Δ in its normal form. Hence, the length of α in terms of atoms (σ_i's) is at most $l\left(\frac{n(n-1)}{2} - 1\right)$. Now, from Theorem 2 we know that the smallest positive conjugator to a rigid braid is obtained by iterated cyclic sliding. Since at every iteration the conjugating element gets bigger, if we are in the generic case we must obtain a rigid element in at most $l\left(\frac{n(n-1)}{2} - 1\right)$ iterations, as we claimed.

If the braid y obtained after the loop in lines 2-5 is rigid, as the algorithms stores the conjugating elements for cyclic sliding at each iteration, we will have a braid α such that $\alpha^{-1} x \alpha = y$.

Now the algorithm checks whether $USS(y)$ is minimal (the generic case we are interested in), as explained in Remark 1, checking whether the minimal simple elements for y are precisely $\iota(y)$ and $\partial(\varphi(y))$.

In general, it is not known how fast it is to compute the minimal simple elements for a given arbitrary braid y. But if y is rigid, one can easily find the minimal simple elements for y. We know that every such element must be a prefix of either $\iota(y)$ or $\partial(\varphi(y))$. For every generator σ_i, one can consider $\sigma_i^{-1} y \sigma_i$ and apply iterated cyclic sliding to it, until it becomes rigid. The obtained conjugating element is the smallest conjugating element from y to a rigid braid, having σ_i as a prefix. We do this for all σ_i which are prefixes of $\iota(y)$, and either we find a conjugating element which is a proper prefix of $\iota(y)$ (in which case $\iota(y)$ is

not minimal), or we have shown that $\iota(y)$ is minimal. Then we do the same for all generators which are prefixes of $\partial(\varphi(y))$. The number of iterations in each case is bounded by the length of $\iota(y)$ (resp. $\partial(\varphi(y))$), which are simple elements, while the total number of generators is $n-1$. So the total number of cyclic slidings used to check whether $\iota(y)$ and $\partial(\varphi(y))$ are minimal (and hence whether $USS(y)$ is minimal) is $O(n^3)$.

If $USS(y)$ is not minimal, we are not in the generic case stated in Theorem 4, hence the algorithm in [7] is applied. Otherwise, we are in one of the situations described in Propositions 3–5. The rest of the algorithm just applies these propositions together with Proposition 2: after decomposing y in the form $y = v^c w^d$, it checks whether both c and d are multiples of k. If this is the case, then $v^{\frac{c}{k}} w^{\frac{d}{k}}$ is the (unique) k-th root of y, and since $x = \alpha y \alpha^{-1}$, it follows that $\alpha v^{\frac{c}{k}} w^{\frac{d}{k}} \alpha^{-1}$ is the desired k-th root of x; otherwise, the algorithm returns the sentence "A k-th root does not exist".

We study now the complexity of our algorithm, assuming that we are in the generic case in which $USS(x)$ is minimal, and we can quickly conjugate x to a rigid braid. Computing the complement or applying τ to a simple element is $O(n)$, and computing $s \wedge t$ for two simple elements s and t is $O(n \log n)$ ([13], Proposition 9.5.1). Starting with an element y in left normal form, computing $\mathfrak{s}(y)$ consists of computing a complement ($\partial(\varphi(y))$), a meet ($\iota(x) \wedge \partial(\varphi(x))$) and the normal form of the conjugate of y by a simple element of length at most l (which is $O(ln \log n)$). Hence the total complexity of applying a cyclic sliding is $O(ln \log n)$.

The first loop (lines 2–5) is repeated $O(ln^2)$ times, checking the condition takes $O(n \log n)$ and the body of the loop takes $O(ln \log n)$. Hence the total complexity of the loop in lines 2–5 is $O(l^2 n^3 \log n)$.

The "If" statement in lines 6–7 is negligible compared with the previous "while" loop.

Next, in lines 8–9 the algorithm checks whether $\iota(y)$ and $\partial(\varphi(y))$ are minimal, for the rigid element y. By the arguments above, this applies $O(n^3)$ cyclic slidings, hence the total complexity of this step is $O(ln^4 \log n)$.

In line 11 and in the loop in lines 12–15, some cyclings are applied. Since the involved braids are rigid of canonical length at most l, and cycling is just a cyclic permutation of the factors with a possible application of τ to a simple element, this final part of the algorithm is negligible with respect to the previous one.

Therefore, the generic-case complexity of Algorithm 1 is $O(l(l+n)n^3 \log n)$. □

Remark 2. *Although the integers p and k are part of the input, the computed complexity does not involve them, as treating with these integers is usually negligible, in reasonable examples, with respect to the calculated complexity. If p is really big, one should take into account the number* $\log p$. *The case of k is somehow different, as one would have a positive answer only if k is a divisor of the integers c and d (with $d \neq 0$), which are $O(p+l)$, so it makes no sense to ask for a k-th root of x, in the generic case, if k is too big compared with p and l.*

Algorithm 1: Find a k-th root of a braid x.

Input : A braid $x \in \mathbb{B}_n$ given in left normal form, and an integer $k > 1$.
Output: A braid $a \in \mathbb{B}_n$ such that $a^k = x$, or the message "*A k-th root does not exist.*".

$y := x$; $l = \ell(x)$; $\alpha = 1 \in \mathbb{B}_n$; $r = 0 \in \mathbb{Z}$;
while $\iota(y) \wedge \partial(\varphi(y)) \neq 1$ **and** $r < l\left(\frac{n(n-1)}{2} - 1\right)$ **do**
  $\alpha := \alpha\, \mathfrak{p}(y)$;
  $y := \mathfrak{s}(y)$;
  $r := r + 1$;
if $\iota(y) \wedge \partial(\varphi(y)) \neq 1$ **then**
  y is not rigid. Apply the algorithm in [7];
else if $\{$*Minimal simple elements for* $y\} \neq \{\iota(y), \partial(\varphi(y))\}$ **then**
  $USS(y)$ is not minimal. Apply the algorithm in [7];
else
  $y' := \tau(y)$; $z := \mathbf{c}(y)$; $PC := \iota(y) \in \mathbb{B}_n$; $t := 1 \in \mathbb{Z}$; $p := \inf(y)$; $l := \ell(y)$; selfConjugateOrbit $:= 0$;
  while $z \neq y$ **and** $z \neq y'$ **do**
    $PC := PC\, \iota(z)$;
    $z := \mathbf{c}(z)$;
    $t := t + 1$;
  if $z = y'$ **then**
    selfConjugateOrbit $:= 1$;
  if selfConjugateOrbit $= 0$ **then**
    $c := p/2$;
    $d := l/t$;
    if $k|c$ **and** $k|d$ **then**
      $v := \Delta^2$;
      $w := PC$;
      return $\alpha v^{\frac{c}{k}} w^{\frac{d}{k}} \alpha^{-1}$;
    else
      return "*A k-th root does not exist.*";
  else if selfConjugateOrbit $= 1$ **and** $y = y'$ **then**
    $c := p$;
    $d := l/t$;
    if $k|c$ **and** $k|d$ **then**
      $v := \Delta$;
      $w := PC$;
      return $\alpha v^{\frac{c}{k}} w^{\frac{d}{k}} \alpha^{-1}$;
    else
      return "*A k-th root does not exist.*";
  else if selfConjugateOrbit $= 1$ **and** $y \neq y'$ **then**
    $t := 2t$;
    $c := \frac{pt+2l}{2t}$;
    $d := \frac{2l}{t}$;
    if $k|c$ **and** $k|d$ **then**
      $v := \Delta$;
      $w := PC\, \Delta^{-1}$;
      return $\alpha v^{\frac{c}{k}} w^{\frac{d}{k}} \alpha^{-1}$;
    else
      return "*A k-th root does not exist.*";

Author Contributions: Investigation and writing: M.C., J.G.-M. and M.S.

Funding: Authors partially supported by the Spanish research project MTM2016-76453-C2-1-P and FEDER. First author was also supported by EPSRC New Investigator Award EP/S010963/1. Third author was also supported by the Basque Government grant IT974-16 and Centro de Estudios Avanzados en Física, Matemáticas y Computación de la Universidad de Huelva.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dehornoy, P. Braid-based cryptography. In *Group Theory, Statistics, and Cryptography*; Volume 360 of Contemporary Mathematics; American Mathematical Society: Providence, RI, USA, 2004; pp. 5–33.
2. Anshel, I.; Anshel, M.; Goldfeld, D. An algebraic method for public-key cryptography. *Math. Res. Lett.* **1999**, *6*, 287–291. [CrossRef]
3. Ko, K.H.; Lee, S.J.; Cheon, J.H.; Han, J.W.; Kang, J.; Park, C. New public-key cryptosystem using braid groups. In *Advances in Cryptology — CRYPTO 2000*; Bellare, M., Ed.; Springer: Berlin/Heidelberg, Germany, 2000; pp. 166–183.
4. Gebhardt, V. A new approach to the conjugacy problem in Garside groups. *J. Algebra* **2005**, *292*, 282–302. [CrossRef]
5. Gebhardt, V.; González-Meneses, J. Solving the conjugacy problem in Garside groups by cyclic sliding. *J. Symb. Comput.* **2010**, *45*, 629–656. [CrossRef]
6. Gebhardt, V.; González-Meneses, J. The cyclic sliding operation in Garside groups. *Math. Z.* **2010**, *265*, 85–114. [CrossRef]
7. Lee, S.-J. Garside groups are strongly translation discrete. *J. Algebra* **2007**, *309*, 594–609. [CrossRef]
8. Sibert, H. Extraction of roots in Garside groups. *Comm. Algebra* **2002**, *30*, 2915–2927. [CrossRef]
9. Dehornoy, P. *Foundations of Garside Theory*; Volume 22 of *EMS Tracts in Mathematics*; Digne, F., Godelle, E., Krammer, D., Michel, J., Eds.; European Mathematical Society (EMS): Zürich, Switzerland, 2015.
10. Artin, E. Theory of Braids. *Ann. Math.* **1947**, *48*, 101–126. [CrossRef]
11. Chow, W.-L. On the algebraical braid group. *Ann. Math.* **1948**, *49*, 654–658. [CrossRef]
12. Elrifai, E.A.; Morton, H.R. Algorithms for positive braids. *Q. J. Math.* **1994**, *45*, 479–497. [CrossRef]
13. Epstein, D.A.; Cannon, J.W.; Holt, D.F.; Levy, S.V.; Paterson, M.S.; Thurston, W.P. *Word Processing in Groups*; A. K. Peters, Ltd.: Natick, MA, USA, 1992.
14. Birman, J.S.; Ko, K.H.; Lee, S.J. The Infimum, Supremum, and Geodesic Length of a Braid Conjugacy Class. *Adv. Math.* **2001**, *164*, 41–56. [CrossRef]
15. Birman, J.S.; Gebhardt, V.; González-Meneses, J. Conjugacy in Garside groups. I. Cyclings, powers and rigidity. *Groups Geom. Dyn.* **2007**, *1*, 221–279. [CrossRef]
16. Birman, J.S.; Gebhardt, V.; González-Meneses, J. Conjugacy in Garside groups II: Structure of the ultra summit set. *Groups Geom. Dyn.* **2008**, *2*, 13–61. [CrossRef]
17. González-Meneses, J.; Valladares, D. On the centralizer of generic braids. *J. Group Theory* **2018**, *21*, 973–1000. [CrossRef]
18. Charney, R. Artin groups of finite type are biautomatic. *Math. Ann.* **1992**, *292*, 671–683. [CrossRef]
19. Caruso, S.; Wiest, B. On the genericity of pseudo-Anosov braids II: Conjugations to rigid braids. *Groups Geom. Dyn.* **2017**, *11*, 549–565. [CrossRef]
20. González-Meneses, J. The n-th root of a braid is unique up to conjugacy. *Algebraic Geom. Topol.* **2003**, *3*, 1103–1118. [CrossRef]